Document No. 0580-001
17 March 1989

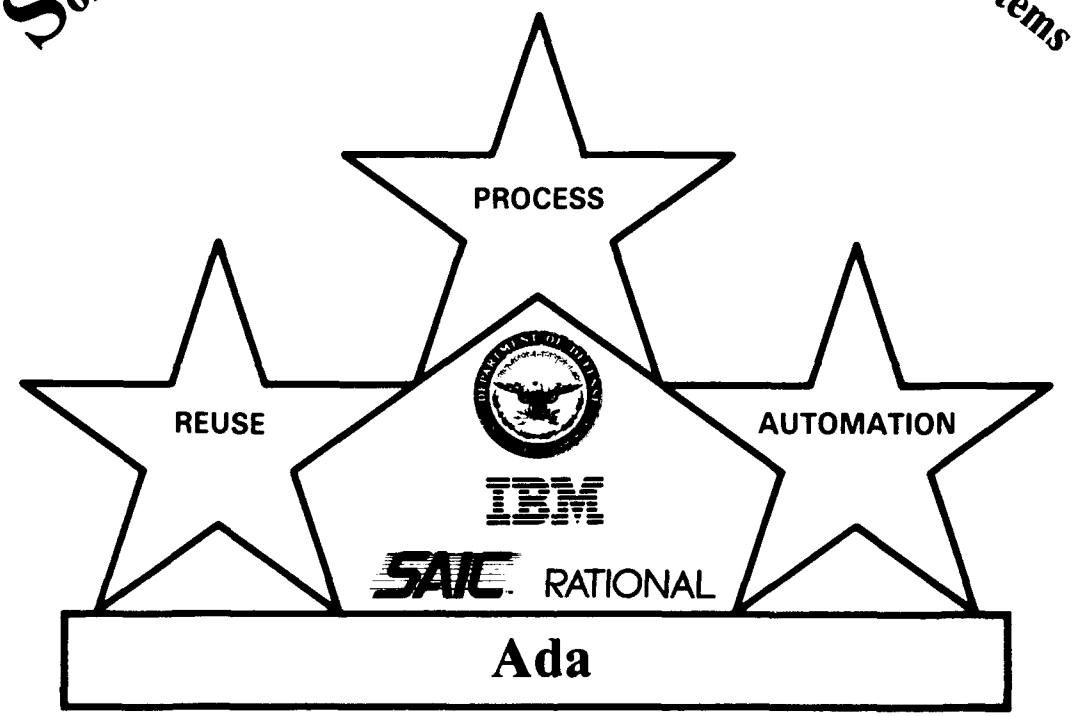
(2)



# Foundation Tools Guidance & Education

for the

## Software Technology for Adaptable Reliable Systems




Contract No. F19628-88-D-0032
CDRL Sequence No. 0580

17 March 1989

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | March 17, 1989 | Final |

**4. TITLE AND SUBTITLE**
Foundation Tools Guidance and Education

**5. FUNDING NUMBERS**
C: F19628-88-D-0032

**6. AUTHOR(S)**

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
IBM Federal Sector Division
800 N. Frederick Avenue
Gaithersburg, MD 20879

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
CDRL Sequence No. 0580

**11. SUPPLEMENTARY NOTES**

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** (Maximum 200 words)

A description of the software tools created during the STARS Foundation work. There are 38 tools, grouped into the following categories:

- Operating systems,
- Data base management systems,
- User interfaces,
- Command languages,
- Graphics,
- Text processing,
- Networks and communications,
- Runtime support,
- Planning and optimization,
- Design, integration and test,
- Reusability assistance, and
- Other tools.

**14. SUBJECT TERMS**
STARS, STARS Foundation, software tools, operating systems, data base management systems, user interfaces, graphics, text processing, computer networks, software reuse

**15. NUMBER OF PAGES**
32

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |

# Foundation Tools Guidance & Education
## for the
# Software Technology for Adaptable, Reliable Systems (STARS) Program

## Contract No. F19628-88-D-0032

## CDRL Sequence No. 0580


## 17 March 1989



**Prepared for:**

Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

**Prepared by:**

IBM Systems Integration Division
18100 Frederick Pike
Gaithersburg, MD 20879

# Q13: Ada Foundations

Contract Data Requirement CDRL 0580, Type A005(B)
Data Item Description DI-H-30255
Contract No. F19628-88-D-0032

Technical Report: Foundation Tools Guidance and Education

17 March 1989

Prepared for:

Electronic Systems Division (PKG-1)
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

Science Applications International Corporation
Innovative Technology Group
Science Technology & Software Operation
Ada Software Division
311 Park Place Boulevard, Suite 360
Clearwater, FL 34619

# TABLE OF CONTENTS

1. INTRODUCTION 1
2. OPERATING SYSTEMS 2
3. DATABASE MANAGEMENT SYSTEMS 4
4. USER INTERFACES 6
5. COMMAND LANGUAGES 7
6. GRAPHICS 8
7. TEXT PROCESSING 9
8. NETWORKS AND COMMUNICATIONS 11
9. RUNTIME SUPPORT 12
10. PLANNING AND OPTIMIZATION 14
11. DESIGN, INTEGRATION, AND TEST 16
12. REUSABILITY ASSISTANCE 18
13. OTHER TOOLS 20
14. OVERVIEW OF FOUNDATION TOOLS 22
15. APPENDICES 24
15.1 FOUNDATION TOOLS DIRECTORY STRUCTURE 24

# 1. INTRODUCTION

## 1.1 Purpose

This contract delivery requirement is intended to provide initial insight into the STARS (Software Technology for Adaptable and Reliable Systems) foundation work. The report addresses paragraph 3.3.2.5 of the Statement of Work (SOW) for the foundation tools task (Q13). Paragraph 3.3.2.5 is concerned with guidance and education of the STARS foundation effort. It focuses on creating an outline of the foundation tool components and their capabilities. In addition, suggestions and recommendations for possible reuse are discussed.

## 1.2 Scope

The STARS foundation tools are located on the STARS repository computer under the directory ADA$:[REPOSITORY.STARS_FOUNDATION]. Each tool is given a unique directory under [.STARS_FOUNDATION] and is organized in the format provided by the contractor.

The foundation work is divided into 12 categories, and each is detailed in a separate section. These categories include:

- Operating Systems
- Database Management Systems
- User Interfaces
- Command Languages
- Graphics
- Text Processing
- Networks and Communications
- Runtime Support
- Planning and Optimization
- Design, Integration and Test
- Reusability Assistance
- Other Tools

Within each section, the functionality and the components of each tool are described. The purpose and intent of the foundation tool are discussed and, when possible, any machine dependencies are noted. System-level documentation is the main source of information for each tool. Where this documentation is not available, a scan of the source code and/or a review of the contractor's SOW is referenced. The environment intended for each tool and the Ada compilers used in their development are also provided. Tools containing possible candidates for reuse are singled out and, when possible, those components are listed. Finally, an overview of the source code, with emphasis on internal documentation, is performed. A summary and an overall review of the foundation tools are given in the last section titled, "Overview of Foundation Tools."

# 2. OPERATING SYSTEMS

This section contains all of the STARS foundation work dealing with operating systems work. The systems include a method of porting file names between different environments, a set of packages for tailoring a runtime environment, and a set of packages for managing streams of data.

## 2.1 Universal File Names

The Universal File Names (UFN) system is written by Integrated Software, Inc. It consists of a set of packages designed to manage file names between several operating systems. The three operating systems which are supported are VAX/VMS, DOS, and UNIX. The system provides an abstract file name which maps to each of the three operating systems and increases the portability of the application software.

The source code is written in Ada, and a driver for a demo is provided with the code. For the most part, the source is lightly documented. It maps all VAX file names to UFN in a 1:1 ratio. The DOS and UNIX names are mapped to a four-digit hash value and the first four characters of the given name. The names are then stored in a Direct_IO file. A utility that converts the Direct_IO file to Text_IO is provided for increased portability.

System-level documentation for the UFN system may be found in the "ufn_2_0.man" file in the repository under the [.INTEGSOFT] directory. All of the source code is in the "ufn_2_0.ada" file.

## 2.2 Tailorable Ada Runtime Environment

The Tailorable Ada Runtime Environment (TARTE) is a collection of packages written by Integrated Software, Inc. It is designed to provide user-tailorable, realtime executive services independent of the standard Ada runtime environment. The standard configuration supports a variety of scheduling schemes. These include coroutines, nonpreemptive priority-based scheduling, and preemptive priority-based scheduling. Priorities are dynamic. A resource abstraction, with binary semaphore operations, is provided. Among other options, the TARTE supports Rate-Monotonic scheduling.

The main function of the TARTE is to provide the kind of user control over scheduling and resource allocation that is needed for realtime applications but is not provided by standard Ada. Specific Ada implementations may also provide such capabilities, but an application that relies on implementation-specific scheduling features is not likely to be portable. The TARTE provides necessary scheduling features in an implementation-independent form, thereby aiding portability and software reuse. The TARTE isolates machine and compiler dependencies in a single package body, named "Machine", that may be rewritten by the user when porting is required.

The TARTE is also intended to be adequate as a foundation for implementation of the full standard Ada tasking model, with extensions for realtime applications and possibly also for implementation of a full realtime operating system.

System-level documentation for the TARTE system may be found in the "tarte_2_0.man" file in the repository under the [.INTEGSOFT] directory. All of the source code is in the "tarte_2_0.ada" file.

## 2.3 General Embedded Systems Specification

The General Embedded Systems Specification (GENESYS) system is written by Aerojet Electro Systems/Intermetrics. At the time of this delivery no source code or documentation has been provided.

## 2.4 Stream Data Types For Ada

The stream management system was written by Computer Technology Associates, Inc. The software contains a reusable set of generic packages for the creation and manipulation of stream objects. Conceptually, streams may be viewed as a generalization of sequential data structures such as arrays, linked lists, and sequential files. Any sequence or flow of a data type and manipulations on that flow can be represented by a stream.

The streams system consists of two packages for creating and manipulating streams. These two packages, Safe_Streams and Free_Streams, are identical as far as the application interface is concerned. They differ in that the procedures in the Safe_Streams package are reentrant while the procedures in the Free_Streams package are not.

The streams software contains a linked list package and a random number generator. These packages may have some reuse potential. There are several test routines and some demonstration routines provided with the code. On the whole the code is well documented. It incorporates good type checking and error handling.

System-level documentation for the Ada streams system may be found in the repository under the [.CTA3_0432] directory in the files "final.doc" and "stream.doc".

# 3. DATABASE MANAGEMENT SYSTEMS

This section contains the database managing software. This includes packages to manage databases and also to interface with existing systems. There is a report generating system ARPS; an Ada/SQL binding to PC ORACLE; and a CICS interface in Ada for IBM systems.

## 3.1 Ada Report Production System

The Ada Report Production System (ARPS) was written by SAIC Comsystems and is an Ada-driven report writer for data stored in a relational database. The Ada/SQL interface that is provided is specific to PC ORACLE. To execute the ARPS for another environment, an Ada/SQL interface for that environment must be created.

ARPS provides the ability to specify and automatically generate reports from a relational database. Reports are created, maintained, and edited using an interactive report specification tool which provides for selection and specification of standard report formatting features and for database extraction using the Ada/SQL DBMS interface. The report specification developed using this interactive tool is an Ada specification which defines, in Ada syntax, the format and content of the desired report. The Ada report specification is processed by the ARPS code generator to produce Ada source code (packages) implementing the report formats and providing procedure calls and interfaces to the Ada/SQL DML for eventual report production.

The ARPS software contains packages that provide windowing utilities, menu utilities, calendar utilities, a parsing utility, and a queue package. These should be considered possible candidates for reuse. The specifications are well documented.

System-level documentation for the ARPS is provided by the files "readme.txt" and "userman.txt" located in the [.SAIC_COMSYS] directory on the repository.

## 3.2 DBMS in/for Supported Information Systems Management

This set of code is a prototype DBMS written in Ada for Ada. The system, written by the Stanford Research Institute, consists of a demo which attempts to show that a DBMS may be written in Ada. The documentation throughout the code is light and Ada's typing and abstraction mechanisms are not thoroughly implemented. The system contains a set of buffer utility packages as well as some relational algebra packages. These packages are implementation-dependent which limits their reuse potential.

There is a "readme.txt" and "port.dat" file located in the repository under the [.SRI] directory. These files do not provide much insight into the structure and functionality of the system. More information may be extracted by looking at the code and reading the SOW.

## 3.3 Prototype Binding of ANSI Standard SQL/Ada

The ANSI standard of SQL/Ada is written by Lockheed Missiles and Space. To date we have not received any source code. According to the SOW, the system provides a standard interface from an application program written in Ada to a commercially available DBMS that uses ANSI standard SQL.

## 3.4 Ada/SQL Test Data Generator

The Ada/SQL Test Data Generator is written by Grumman Data Systems. This system

generates test data for Ada/SQL interfaces according to user-supplied data attributes. The internal documentation is light and makes little use of Ada's typing and abstraction mechanisms. System-level documentation is not provided, and the source code may be found in the [.GRUMMAN] directory.

### 3.5 Transparent Sequential Input/Output

The Transparent Sequential Input/Output (TSIO) is written by Computer Sciences Corporation. It provides an alternative to Ada's predefined Sequential_IO package. TSIO's main point of interest is that the files that are produced are portable where as Sequential_IO's files are not. The system has been compiled on several machines including a DEC VAX, a Rational and an IBM PC/AT. Before TSIO will execute correctly, the user environment must be set up and an Ada types library created. At a minimum, setting up the user environment requires defining the logical name ADA$TYPES_LIBRARY at the process, group, or system level. If the name is defined at more than one level, the TSIO software uses the lowest level at which the logical name is defined.

System-level documentation for the TSIO may be found in the "read_me.lis" file in the repository under the [.CSC] directory. The system consists of several driver programs that create the type structures and perform the input and output.

### 3.6 Ada Interface to CICS and SQL

The CICS interface is written by Intermetrics. The system provides explicit access to CICS services. These packages consist primarily of code which passes control to the standard CICS assembly language command-level interface. For this reason, the Ada procedures will, in general, do whatever the corresponding command-level instruction would do with similar parameters.

In addition to a CICS interface, an SQL module compiler is provided. This tool provides a way for application utilities, written in SQL module language, to be translated into Ada packages that can be used when writing Ada applications for SQL. The Ada application programs can perform database queries and database manipulation in SQL on Datacom/DB databases. The SQL/Ada Module Compiler generates a compilable Ada package specification and body from one SQL module. Ada applications can then "with" this package to gain access to databases that have an SQL/Ada interface.

The code contains approximately 41 assembly language files which limit the portability to IBM machines. It was compiled using the Intermetrics MVS Ada compiler. Overall the code is lightly documented and displays several coding formats.

System-level documentation for the CICS and the SQL interfaces is located in the files "man.doc", "sql.doc", and "hostdep.dat". They may be found in the repository under the [.INTERCICSQL2] directory.

# 4. USER INTERFACES

## 4.1 Virtual CAI Interfaces

The Virtual CAI Interfaces system is written by AETech. It defines a computer-aided instruction system. Resources provided include keyboard and file routines, I/O routines for a mouse, I/O routines for a video disk, graphics-mode display packages, and asynchronous communication packages.

The code has been developed on an Alsys compiler for the IBM PC/AT. The package specifications are well documented. System-level documentation is not provided, and the source code may be found in the [.AETECH_CAI] and the [AETECH_VI] directories.

# 5. COMMAND LANGUAGES

## 5.1 Ada Command Environment

The Ada Command Environment (ACE) is written by Unisys. The ACE is an interactive, object oriented command language environment for Ada development. It is modeled after existing interactive programming environments, such as Smalltalk and Interlisp. ACE supports many programming-in-the-large techniques that are key elements of the Ada language.

For the initial environment, ACE provides a basic set of Ada objects and operations. These objects and operations are encapsulated as Abstract Data Types (ADTs) and implemented as a set of Ada packages. Upon initialization of ACE, the basic set of ADTs is assimilated into a base environment for the user. This base environment includes packages that are necessary for the interpretation of Ada statements (such as the Ada package "Standard") as well as packages that provide operations typically performed by a user when interacting with the underlying operating system.

System-level documentation is provided in the files "manual.txt" and "porting.txt" located in the directory [UNISYS_ACE3.DOCS]. More documentation may be found in the "bugs.txt", "pc.txt", "unix.txt", "verdix.txt", and "howtouse.txt" files located in the [UNISYS_ACE3.READMES] directory. The source code is well documented and follows a uniform programming format. The ACE prototype is operational in the Sun-3 and Unisys PW2/500 (IBM PC/AT-compatible) environments.

# 6. GRAPHICS

## 6.1 Ada Language Interface to the X-Window System

The Ada interface to the X-Window system is written by Science Applications International Corporation (SAIC). The system implements the X-Window system using Ada. X-Windows is a network transparent windowing system developed at MIT for several operating environments. The Ada implementation is intended to support larger applications that wish to incorporate X-Windows into their system.

System-level documentation is not provided but there is a "readme." file in the [.SAIC3] directory which comments on installation of the software. The software is intended to be portable. There are seven files written in C that must be compiled into a library and bound to the X-Window system. The system was developed using the Telesoft, Alsys, and Verdix Ada compilers. Files necessary for porting the system to these environments have been provided.

## 6.2 Graphics Kernel System in Ada

The Graphics Kernel System (GKS) is a terminal graphics system implemented in Ada by Software Technology, Inc. It is not known whether this particular implementation follows the ANSI standard. GKS provides packages for trigonometric functions and a square root function which may be reusable. The internal documentation is minimal and there is no indication as to how to adapt the system to a particular machine. There are no user manuals and no system-level documentation.

# 7. TEXT PROCESSING

The text processing section contains all of the systems involved in processing or generating text. Among the systems, there is a set of reusable components, an SGML parser and text formatting system, and the STARS Editor.

## 7.1 Terminal Interface Package and Building Blocks Packages Upgrade

The Terminal Interface Package and Building Blocks Upgrade is written by the Ada Software division of SAIC. It consists of reusable components that were first written for the PC/AT environment and then ported to the VAX/VMS environment. Components include balanced tree, linked list, stack, symbol table, calendar utilities, and utilities for both static and dynamic strings. The packages are intended to aid in the development of larger software projects by eliminating the need to rewrite elementary modules. Included in the delivery are packages for terminal and keyboard input and output.

The SAIC building blocks have been developed using the Alsys and the VAX/VMS Ada compilers. Most packages are stand-alone components, but some require the services of an operating system and CRT interface which is provided. There are many test and demonstration routines for the components, and the code is well documented. The documentation and source code may be found in the repository under the [.SAICBB3] directory.

## 7.2 SAIC Text Editor Upgrade in Ada

The SAIC Text Editor Upgrade is written by the Ada Software Division of SAIC. This is the STARS Foundation editor developed for NRL. It provides multi-window facilities and is designed to run on the VAX/VMS or PC/AT environment. The editor provides a keyboard template similar to WordStar and a menu-like command format for text editing.

The editor's features include a hierarchical on-line help system, split-screen editing, multiple buffer editing, an annotation system, and syntax-directed editing. The syntax-directed feature includes the Ada language and selected Standard Generalized Markup Language (SGML) document types. These are similar in function to the Language Sensitive Editor (LSE) written by DEC.

The source code is well documented and follows a uniform programming format. Some drawbacks include occasional loss of video presentation and the requirement of extended memory to operate in the PC/AT environment. A user's manual may be found in the [.SAIC_SGML] directory, and the source code may be found in the [.SAIC_TP2] directory.

## 7.3 Standard Generalized Markup Language (SGML) Implementation in Ada

The SGML parser is written by the the Ada Software Division of SAIC. It is a partial implementation of ISO 8879 which describes a general markup language for documents. The parser accepts as input a document which is marked using SGML tags. It produces a fully expanded document with the tags and text separated and expanded. The tool includes a Text Composition System (TCS) which takes the fully expanded file produced by the parser and creates a PostScript file that is ready for printing.

The parser comes with a front-end Document Type Definition (DTD) converter which parses a valid document BNF (according to ISO 8879). This BNF describes the format of the document. An error encountered in the document will cause the parser to terminate by informing

the user that an incorrect sequence of text and markup has been found. A DTD must be written for each document type the user needs to process.

The TCS requires that the user code (in Ada) a document-specific driver. This driver consists of package which contains a set of subroutines that formats the document according to the user's specifications. An extensive set of routines is provided to aid the user in writing the driver. Three examples are provided for the user to reference when writing the TCS driver.

System level documentation for the code may be found in the files "stcs.sgm", "tcs.sgm", "stcs.ps", and "tcs.ps" within the directory [.SAIC_SGML]. The *.ps files are PostScript files. The parser is written for the VAX/VMS environment and for the PC/AT environment. Some drawbacks include slow execution speed and the need for extended memory to operate on the PC. The SGML system is well documented and follows a uniform programming format.

# 8. NETWORKS AND COMMUNICATIONS

## 8.1 Network Protocol

The Network Protocol system is written by Westinghouse Electric. The Network Protocol is a set of Ada packages which implement two well-known transfer protocols. The two protocols which are implemented are X-Modem and Kermit. The protocol system is implemented as a group of generic packages. Both file transfer protocols are implemented as generics so that the process for using either one is exactly the same.

Network Protocol implements the elements of both the X-Modem and Kermit file transfer protocols which are considered to be standard. The functions provided are Send and Receive. The fact that there are only two functions instead of four is a result of the use of generics to implement both file transfer protocols. The invocation of either the Send or Receive function will be performed in the same way regardless of which file transfer protocol has been instantiated.

The use of Network Protocol is dependent on the user adapting the application to fit the target environment. Since Network Protocol is not a stand-alone program, certain procedures must be written before it is executable. A definition for the type "byte" must be supplied, along with procedures that provide the ability to read and write to the RS-232 port (or whatever is being used to "transfer" the data). Procedures that access a file in the medium that the user desires are also needed. Once these conditions have been met, it is important to remember that the protocol selected must be available on the target machine.

System level documentation is located in the "aaaread.me" and the "users_manual.txt" files in the [.WESTING_NP] directory. There are two demonstration drivers provided to help in the understanding of the system. The specifications contain some light documentation. The Network Protocol system is implemented for both the VAX/VMS and the Rational R1000 environments.

# 9. RUNTIME SUPPORT

## 9.1 Transparent Distributed Ada Runtime Support

The Transparent Distributed Ada Runtime Support is written by the ESL Corporation. It is a runtime support environment. The system consists of a code conversion of an old C implementation. No documentation is provided, and it appears as though this is a UNIX-based system. Some comments may be found in the "$readme." files located in the [.ESL.68K] and the [.ESL.SUN] directories.

## 9.2 Secure File Transfer Program

The Secure File Transfer Program is written by the BDM Corporation. It provides a means of transferring sensitive information from one computer system to another. This system implements Transmission Control Protocol (TCP) and conforms to RFC:793. TCP provides reliable communication between pairs of processes in logically distinct hosts on networks and sets of interconnected networks. The TCP resides in the transport layer of the DOD Internet Model. It encapsulates messages received from the utility layer protocols which reside directly above, and passes the packet to, the Internet layer (communicates with the Internet Protocol(IP)). The TCP supports the following functions: connection-oriented, reliable data transfer, ordered data transfer delivery, full-duplex and flow control.

TCP communicates with both upper-layer and lower-layer protocols by means of response and request primitives. A multiplexing mechanism is built into TCP to support multiple upper-layer Protocol (ULP) processes by a single TCP process. Messages passed into TCP from ULPs are buffered and sent when the appropriate size is adequate for efficient transfer or internal processing warrants communication. All communication between entities is synchronized and flow control maintained by TCP.

System-level documentation may be found in the "tcp.man" and the "sftpuser.man" files located in the [.BDM] directory. The code is written for the VAX/VMS and the Telesoft WICAT environments. The specification source code contains some light documentation.

## 9.3 Ada Runtime Support for Complex Time-Critical Embedded Applications

This runtime support system is written by Advanced System Technologies. It is an intertask communication and shared resource control system. The Ada runtime services provided by the system are intended to satisfy a majority of the requirements for high-level control and synchronization functionality. The services required fall into the following six categories: buffered intertask communication, shared data access management, timed data buffer, periodic tasks, task scheduling, and task pool management.

System-level documentation may be found in the "user_manual.txt" file located in the [.ADSYSTECH2] directory. The source code is well written, well documented, and follows a uniform coding style. The system has been developed for the VAX/VMS and the PC/AT environments. It exhibits potential for reuse in simulation and realtime applications.

## 9.4 Synthesis of Large Systems with a Versatile Server Package

The Versatile Server Package is written by Computer Technology Associates. This package

enables a designer to specify a new software system as a composition of several subsystems. In addition, the designer can express, in Ada, the data dictionary associated with the data flows that connect the subsystem. The software described here can take this Ada-based description and use it to generate Ada code that implements the interface tasks required by the subsystems. Alternative code generators are provided so that the interface can be internal to a single hardware host or external between two or more hosts.

System-level documentation may be found in the "uguide.lst", "pguide.lst", and "dguide.lst" files located in the [.CTA4_0433] directory. These manuals provide a description of the system as well as specific help for application programmers. A thorough demonstration program is included with the system. The source code is very well documented and follows a well defined, uniform programming format. The code has been developed for the PC/AT environment using the Alsys Ada compiler.

### 9.5 Ada Remote Procedure Call

The Ada Remote Procedure Call (RPC) system is written by Software Architecture and Engineering. The RPC tool kit is a collection of facilities defining a layered hierarchy of communication protocols that enable an Ada programmer to construct a distributed implementation of an Ada application (subsystem) using the remote procedure call paradigm. The tool kit is implemented as a collection of Ada packages, generic packages, and a generic subprogram.

The Ada RPC tool kit is organized into four subsystems of communication protocol abstractions. Each subsystem is represented by a collection of Ada packages that contain the associated type definitions and subprograms to provide the required capability. These subsystems are: presentation protocol, session protocol, representation protocol, and transport protocol.

The presentation, session, and transport protocol subsystems are organized into a linear hierarchy that closely follows the ISO OSI model of a layered internetwork communication protocol. The representation protocol subsystem is utilized in the implementation of the presentation and session protocol subsystems and is also used within user-written code for a particular application (subsystem) to construct instantiation parameters for the facilities in the presentation protocol subsystem.

System-level documentation may found in "rpc-ph1.man" and "rpc-ph1.readme" files located in the [.SA_E] directory. The source code is well documented and adheres to a uniform programming format. It has been designed to operate in the PC/AT and the SUN-3 (UNIX) environments. Some C source code is provided to gain access to a few of the UNIX C network library functions that cannot be bound directly to Ada.

# 10. PLANNING AND OPTIMIZATION

This section contains all of the planning and optimizing tools. Among the systems there is an Ada implementation of several Dijkstra algorithms, and a set of simulation tools (TASKIT). Also included in this section are a set of resource requirements pairing algorithms and a set of reusable allocation optimizing algorithms (Reusable Ada Modules for Resource Allocation).

## 10.1 Optimization and Planning Tools

This system is a collection of Dijkstra's algorithms and is written by Systems Control Technology. The algorithms include Dijkstra's shortest path between nodes, dynamic programming algorithm, terrain masking package, and route retrieval package.

Each of the algorithm implementations contains a demonstration program and a users manual. Specifically, this documentation may be found in "dijkstra_users.man", "dpa_users.man", "masking_users.man", and "route_retrieval_users.man" files. The files are located in separate subdirectories under the [.SCT_OP2] directory. The code is well documented and follows a uniform coding format.

## 10.2 Planning and Optimization Algorithms

This set of optimization algorithms is written by Lockheed. The algorithms implemented include an optimization of resource to requirements pairing and an Ada implementation of resource allocation modules. The resource allocation modules are taken from the existing Lockheed system Reusable Ada Modules for Resource Allocation (RAMORA).

The heart of RAMORA consists of three major reusable software elements inherent in most resource allocation systems; Effectiveness, Criteria and Optimization. The Effectiveness package computes the effectiveness values for each resource/requirement pair. RAMORA's effectiveness estimates come from tables of data based on targets and weapons. These tables are taken from a DOD document called the Joint Munitions Effectiveness Manual (JMEM). JMEM data is used to compute the number of required weapons to inflict a desired level of damage on a specified target. The Criteria package computes criteria values based on output from the Effectiveness package and some system-dependent weighting factors. These criteria values are then used by the Optimization package to optimally pair the entered targets with selected weapons.

System-level documentation may be found in the [.ABSTRACTS] directory in the file "martin.txt". The source code may be found in the [.LOCKHEED_POA2] directory.

## 10.3 Tasking Ada Simulation Kit

The Tasking Ada Simulation Kit (TASKIT) is written by the Data Systems Division of General Dynamics. It consists of a set of tools that enables the use of Ada as the development language for simulations. The system provides the programmer with services common to many types of modeling applications. These functions include an activity manager, keyed linked list manager, math functions package, random number services utility, and resource protection.

The tools are implemented as user-callable procedures and functions and are organized into Ada packages. These packages can be used individually or in any combination to develop models and may even be used for non-simulation applications. TASKIT uses an activity-oriented modeling approach which is similar to discrete event modeling. This activity-oriented world view was chosen so that simulations constructed using TASKIT can take advantage of parallel

processing. The activity orientation can also be used to construct network type simulations.

In addition to the tools for developing simulation models, TASKIT provides a system called SDMS (Simulation Data Management System) which manages input and output data for simulations. It enables the user to enter simulation data in user-defined tables using a spreadsheet format and provides a library of procedures that enable application programs to read and write data to the tables. SDMS is described in Volume II of the user's manual. It is treated separately since it is a stand-alone system and is completely independent of the other TASKIT components.

System-level documentation may be found in the "user_manual_vol_I.doc" and "user_manual_vol_II.doc" files located in the [.GENDYNAM4] directory. The system was developed using the VAX, Alsys, Verdix Harris, and Telesoft/GD Ada compilers. The code is well documented and follows a uniform coding format.

# 11. DESIGN, INTEGRATION, AND TEST

This section contains all of the software for designing and testing. Among the systems there is a test generating tool, an embedded systems debugger, and a configuration management assistant.

## 11.1 Ada Test Support Tool

The Ada Test Support Tool (TST) system is written by Intermetrics. The TST is written in Ada to provide for the testing and analysis of Ada programs. TST provides the user with a compiler-independent and portable tool for testing individual routines or subprograms having visibility within a compilable Ada unit.

TST prompts the user for the input to the program units being tested, checks the validity of the parameters, executes the program using the test values specified by the user, and collects the results of the test. TST then provides a summary of the test execution which shows the subprogram tested, input values, output values, and function results. An optional path analysis report can be included which shows the paths executed and how often each statement in the program was executed.

TST consists of a source instrumenter, a runtime Monitor, a report generator, and an on-line help facility. These are combined into a TST shell providing an integrated environment from which the four components of TST can be executed. Additionally, TST provides an interface to the system level, which enables the user to execute operating system commands from within the shell.

System-level documentation may be found in the "tst.ug" and "read.me" files located in the [.INTER_TST] directory. The TST has been developed using the Alsys and VAX Ada compilers. The code is well documented and follows a uniform coding style. There is an on-line help facility that provides the user with information on syntax and the TST capabilities.

## 11.2 Ada Embedded Systems Debugger

The Ada Embedded Systems Debugger (ESD) is written by Intermetrics. ESD is a symbolic debugger for computer programs implemented in the Ada programming language. Its purpose is to enable the user to locate errors in program logic by providing a "window" into the program's execution through which the user can monitor and, to a certain extent, alter the status and flow of control of the Ada program being tested.

All interfaces between the user's program and ESD are accomplished through source instrumentation. As a result, a portion of ESD actually becomes a part of the user's program when it is executed. Thus, it is subject to the same restrictions, limitations, and constraints as the user's program. For example, constants may be examined but they may not be modified. On the other hand, OUT mode parameters in subprograms may be modified, but they may not be examined.

System-level documentation may be found in the "userguid.esd" file located in the [.INTER_ESD] directory. The code is well documented and follows a uniform coding format. ESD has been designed using the Alsys , VAX, and Meridian Ada compilers.

## 11.3 Configuration Management Assistant for Ada Environments

The Configuration Management Assistant (CMA) is written by Tartan Laboratories. As stated in the SOW, the CMA will administer versions of system components and tools. It allows users to describe configuration families and specific sets of versions. Completeness and consistency checks may be performed on the configuration elements. CMA integrates nested configurations. This allows the tailoring of the configuration of a component to fit the specific needs of that component. At the time of this delivery, no source code or documentation has been provided.

### 11.4 Table Building Generator

The Table Building Generator (TBG) is written by Tartan Laboratories. According to the SOW, the TBG is intended to facilitate preparation of initialized tables of data as part of an application. The tool takes as input a high-level data description along with an ASCII representation of the data. It performs various optimizations, in storing the tabular data, while providing rapid and efficient access. At the time of this delivery, no source code or documentation has been provided.

### 11.5 Parser Builder

The Parser Builder system is written by Westinghouse Electric. This system takes as input BNF grammar for an application programming language and outputs a file containing the tables needed by the parser. Some useful packages include a lexical scanner, date and time utility package, sorting package, hash table package, and list package.

System-level documentation is not provided, but there is an installation message in the "read.me" file located in the [.WESTING_PARSER] directory. The source code has been developed for the VAX/VMS and Rational environments.

# 12. REUSABILITY ASSISTANCE

This section contains software for increasing reusability. Among the systems, there is a composer for generating PDL and an expert system modeled after a Unisys-based knowledge representation system.

## 12.1 Reusability Library Framework

The Reusability Library Framework is written by Unisys. It implements three framework subsystems AdaKNET, AdaTAU, and Gadfly. The Gadfly subsystem has not been delivered. The subsystems are Abstract Data Type implementations of Unisys based knowledge representation systems. AdaKNET is a structured inheritance representation system and AdaTAU is a rule-based inferencer.

AdaKNET is based partly on KL-ONE, and partly on K-NET, Unisys-proprietary structured inheritance system. Use of the AdaKNET ADT requires a thorough understanding of the sometimes complex semantics of structured inheritance networks. AdaKNET is implemented as layered abstract data types (ADTs). Users of AdaKNET need only "with" a single package (package AdaKNETs). AdaKNET provides a programmatic interface to a semantic network model; operations are provided for creating, saving, restoring, manipulating, and examining the structure of AdaKNET instances.

AdaTAU is based on TAU (Think, Ask, Update), a Unisys-proprietary production rule-based system that incorporates an agenda mechanism for directing interaction with a user along with a forward-chaining inference system. Rule base systems provide deductive capability to generate new information based on information already present in the system. Information is stored in the form of facts that are represented in AdaTAU as attribute, value pairs. Rule bases are simply collections of rules, each of which includes a list of facts that must be true to apply the rule, and a list of facts to be concluded when the rule is applied.

System-level documentation for the AdaKNET subsystem may be found in the "user_manual." file in the [.UNISYS_RLF2.ADAKNET] directory. Documentation for the AdaTAU subsystem may be found in the "user_manual." file located in the [.UNISYS_RLF2.ADATAU] directory. The subsystems were developed using the Verdix Ada Development System 5.5 on a Sun-3 network.

## 12.2 Ada Composer

The Ada Composer is written by Intermetrics. The Composer is a graphical design tool that supports the interactive creation of object oriented design diagrams (OODDs) and translates them into compilable Ada program design language (PDL). The Ada Composer supports the reuse of existing software components by using a source to graph converter (STGC) that translates existing Ada reusable components into graphical representations that may be included in OODDs. Reusable component source code may be automatically inserted into the PDL at the user's request. The Ada Composer builds on the graphic Ada designer (GAD) developed by the SYSCON Corporation for the WIS Ada toolset.

The Composer has an interactive and graphical user interface that provides an easy way to compose existing reusable software components with newly created components to design software systems. The generated PDL serves as a compilable top-level design document that may be further refined into a detailed design and then into the final source code. By including reusable component source, the PDL is closer to the final product, thus decreasing software

development time.

System-level documentation may be found in the "users.txt" file located in the [.INTER_COMP] directory. The Composer was developed using the Alsys and VAX Ada compilers. The source code is well documented.

### 12.3 Rapid Storage and Retrieval of Reusable Components (RSR)

The RSR system is written by AETech. To date no source code or documentation has been provided.

# 13. OTHER TOOLS

This is the last section and it contains a collection of systems that do not necessarily fit into the previously mentioned categories. Among them there is a set of packages for developing image processing applications, and a set of pattern recognition packages.

## 13.1 Factory/Maintenance Fault Isolation

The Factory/Maintenance Fault Isolation system is written by ITT Avionics. The system contains packages for reading, reducing, and printing recorded data (taped). It contains very little system level documentation. The documentation files are in an unknown format which is not supported on the repository computer and are therefore unreadable.

## 13.2 Instrumented System Development Software Components

The Instrumented Systems Development is written by AETech. At the time of this delivery no source code or documentation has been provided.

## 13.3 Reusable Image Processing Packages

The Reusable Image Processing Packages are written by Ford Aerospace Corporation. The Reusable Image Processing Packages contract focuses on the preprocessing category. These algorithms are the foundations for more sophisticated image processing algorithms. The packages include histogram operations, point operations, neighborhood operations, and morphological operations.

The image processing packages are intended as building blocks for different imaging applications. They can be used to build more complex image processing functions by combining several of the lower level functions.

System-level documentation may be found in the "finalreport.lis" and "usersguide.lis" files located in the [.FORD4] directory. The source code was developed using the Telesoft VAX and the Alsys PC/AT Ada compilers.

## 13.4 Pattern Recognition

The Pattern recognition packages are written by Westinghouse Electric. The purpose of the pattern recognition system is to classify bit strings using an algorithm proposed by John Holland of the University of Michigan. The system is a learning classification program that can be applied to user-defined domain environments with little or no code modification of the core routines. Bit strings to be classified are called environmental messages. A set of pattern bit strings are used as the rules in the system. The strength or importance of these rules can change to meet the particular domain environment. Interim messages may be generated before the system produces some result.

The pattern recognition system is not intended as a stand alone system. The system is intended to serve as a reusable classification engine to be inserted into a larger application. An example application is provided to show the operation of the pattern recognition system. The application contains routines to perform the initialization required for the particular domain environment and routines to perform post processing once the pattern recognition system has halted. This example may be used to reduce the amount of application specific code that must be designed from the start. These routines have more application dependencies than the Pattern

Recognition System and are not included in the reusable portion since they affect the reusability of the Pattern Recognition System. But they can be reused within the given application domain.

System-level documentation may be found in the "users_manual.lis" and "read.me" files located in the [.WESITNGPR2_2] directory. The system was developed for the VAX/VMS and Rational 1000 environments. The code is lightly documented.

# 14. OVERVIEW OF FOUNDATION TOOLS

## 14.1 Notable Systems

Several of the foundation tools stand out, and should be noted for their more than adequate documentation, coding format, and insightful use of Ada. When a tool is singled out as notable we are not implying that the software is without flaws or that the tool is easily portable (although this may be the case). By notable we are saying the tool stands out from the rest of the foundation systems in this report. The time allotted for this report did not provide for a thorough investigation of all the components for every tool. All of the observations are made by examining the Statement of Work (SOW) for a particular tool, reading system level documentation (where available), and scanning several source code files.

### 14.1.1 Documentation and Style

The Ada Command Environment (ACE) written by Unisys is the first example. The ACE system gives the user an interactive, object oriented, command language environment for Ada development. It provides an excellent range of system level manuals. There is a user manual, a manual listing the existing bugs and several manuals which aid in the porting of the software from a PC, to UNIX, or to a Verdix environment. The source code is well documented and follows a single uniform coding format. By following a single format and style the code is much easier to comprehend and maintain.

The Versatile Server Package written by Computer Technology Associates is another excellent example. This system allows a designer to specify a new software system as a composition of several subsystems. This is an excellent method to follow when engineering large systems. The Versatile Server Package provides a user manual, a programmer's manual, and a demonstration manual. The source code is very well documented and follows a single uniform coding format. The code has been developed on the PC/AT environment using the Alsys Ada compiler.

Finally, The Tasking Ada Simulation Kit (TASKIT), written by General Dynamics, gives the user a set of tools for writing simulations in Ada. This system provides two user's manuals and has been developed on four Ada compilers. A Simulation Data Management System (SDMS) is available and is described in volume two of the user manual. The source code is well documented and follows a single uniform coding format.

### 14.1.2 Candidates for Reuse

The X-Window system written in Ada by SAIC/San Diego is an application of today's most popular windowing technology. The X-Window system originated at MIT as a means of creating a network transparent windowing system. It consists of a set of common routines that manage the terminal and the keyboard. The specifications remain unchanged and the underlying code is written for specific target hardware. The first approach uses the C programming language but the X-Window technology is excellently suited for Ada. The Ada system written by SAIC/San Diego has been developed using three compilers. It is intended to be portable.

The set of Optimization and Planning Tools written by Systems Control Technology is a set of Dijkstra algorithms written in Ada. Each of the algorithms contains a demonstration program and a user manual. The algorithms may be an addition to any set of reusable components or could be the basis for one.

The Embedded Systems Debugger (ESD) written by Intermetrics is a symbolic debugger for programs written in Ada. The notable aspect of this system is that it interfaces with the user's program and actually becomes part of the program. The user can monitor and, to a certain extent, alter the status of the program.

All of the above systems exhibit notable features that help them stand out from the others. This is by no means a comprehensive overview of the foundation tools but rather an introduction into the what is available in the STARS repository under the [.STARS_FOUNDATION] directory. Several of the systems contain very little documentation and this hinders their representation here. Some are incompatible with the VAX/VMS environment and this prevents a fair evaluation of the system. In addition some of the systems are conversions of existing tools. This conversion is not always an improvement and advances the point that a reusable and well engineered software system should be designed with reuse in mind from the start.

# APPENDIX 15.1

## FOUNDATION TOOLS DIRECTORY STRUCTURE

This section gives a graphical representation of the directory structure of the STARS_FOUNDATION directory on the STARS repository. All of the top-level directories (below the [.STARS_FOUNDATION] directory) pertain to a specific foundation contract. The contents of the directories are discussed in greater detail in the previous sections above.

```
Subdirectories below level ADA$:[REPOSITORY]

.STARS_FOUNDATION
.       .ABSTRACTS
.       .ADSYSTECH2
.       .AETECH_CAI
.       .AETECH_VI
.       .AREA
.       .BDM
.       .CSC
.       .CTA3_0432
.       .CTA4_0433
.       .       .REPORT
.       .       .SLIDES
.       .       .SOURCE
.       .ESL
.       .       .ASP
.       .       .       .68K
.       .       .       .       .$5NIMPORTS
.       .       .       .       .$5NLINES
.       .       .       .       .$5NNETS
.       .       .       .       .$5NOBJECTS
.       .       .       .PLAYERS
.       .       .       .68K_BIN
.       .       .       .       .$5NIMPORTS
.       .       .       .       .$5NLINES
.       .       .       .       .$5NNETS
.       .       .       .       .$5NOBJECTS
.       .       .       .SUN
.       .       .IPC
.       .       .       .68K
.       .       .       .       .$S$AVE
.       .       .       .       .DEBUG
.       .       .       .       .       .$5NIMPORTS
.       .       .       .       .       .$5NLINES
.       .       .       .       .       .$5NNETS
.       .       .       .       .       .$5NOBJECTS
.       .       .       .       .GENERIC
.       .       .       .       .HARDWARE
.       .       .       .       .INTERFACE
.       .       .       .       .       .$5NIMPORTS
.       .       .       .       .       .$5NLINES
.       .       .       .       .       .$5NNETS
```

```
.       .       .       .       .       .$5NOBJECTS
.       .       .       .       .SPECIFIC
.       .       .       .       .SUN_DEBUG
.       .       .       .       .       .$5NIMPORTS
.       .       .       .       .       .$5NLINES
.       .       .       .       .       .$5NNETS
.       .       .       .       .       .$5NOBJECTS
.       .       .       .       .SUN_HARDWARE
.       .       .       .       .SUN_INTERFACE
.       .       .       .       .       .$5NIMPORTS
.       .       .       .       .       .$5NLINES
.       .       .       .       .       .$5NNETS
.       .       .       .       .       .$5NOBJECTS
.       .FORD4
.       .GENDYNAM4
.       .GRUMMAN
.       .INTEGSOFT
.       .INTERCICSQL2
.       .INTER_COMP
.       .INTER_ESD
.       .INTER_TST
.       .ITTAVION
.       .       .UATL
.       .       .       .COMMON
.       .       .       .       .COMMON_LIB
.       .       .       .       .PC
.       .       .       .       .VAX
.       .       .       .DISPLAY
.       .       .       .       .CMS
.       .       .       .       .DISPLAY_LIB
.       .       .       .       .PC
.       .       .       .       .PRETTY
.       .       .       .       .       .ACS
.       .       .       .       .SCA
.       .       .       .       .TEMP
.       .       .       .       .VAX
.       .       .       .GIGA_TRONICS
.       .       .       .       .GIGA_LIB
.       .       .       .I1553_IO
.       .       .       .       .I1553_LIB
.       .       .       .PC
        .       .       .       .VAX
.       .       .       .I488_IO
.       .       .       .       .I488_LIB
.       .       .       .       .PC
.       .       .       .       .VAX
.       .       .       .INTERNAL_IO
.       .       .       .       .INT_LIB
.       .       .       .RECORDING
.       .       .       .       .RECORD_LIB
.       .       .       .REDUCTION
.       .       .       .       .PC
```

```
.       .       .       .       .REDUCE_LIB
.       .       .       .       .REDUCE_SCA_LIB
.       .       .       .       .VAX
.       .       .       .STIM_RESPONSE
.       .       .       .       .STIM_RESP_LIB
.       .       .       .       .TEMP
.       .       .       .UATL
.       .       .       .       .COMMON
.       .       .       .       .       .COMMON_LIB
.       .       .       .       .       .PC
.       .       .       .       .       .VAX
.       .       .       .       .DISPLAY
.       .       .       .       .       .DISPLAY_LIB
.       .       .       .       .       .PC
.       .       .       .       .       .VAX
.       .       .       .       .I1553_IO
.       .       .       .       .       .I1553_LIB
.       .       .       .       .       .PC
.       .       .       .       .       .VAX
.       .       .       .       .I488_IO
.       .       .       .       .       .I488_LIB
.       .       .       .       .       .PC
.       .       .       .       .       .VAX
.       .       .       .       .INTERNAL_IO
.       .       .       .       .       .INT_LIB
.       .       .       .       .RECORDING
.       .       .       .       .       .RECORD_LIB
.       .       .       .       .REDUCTION
.       .       .       .       .       .PC
.       .       .       .       .       .REDUCE_LIB
.       .       .       .       .       .VAX
.       .       .       .       .STIM_RESPONSE
.       .       .       .       .       .STIM_RESP_LIB
.       .       .       .       .USER
.       .       .       .       .       .USER_LIB
.       .       .       .USER
.       .       .       .       .USER_LIB
.       .ITTAVIO2
.       .       .UATL
.       .       .       .UATL
.       .       .       .       .BUILDER
.       .       .       .       .COMMON
.       .       .       .       .       .COMMON_LIB
.       .       .       .       .       .PC
.       .       .       .       .       .VAX
.       .       .       .       .DISPLAY
.       .       .       .       .       .PC
.       .       .       .       .       .VAX
.       .       .       .       .DOCUMENTATION
.       .       .       .       .DRIVER
.       .       .       .       .       .GIGA_1018
.       .       .       .       .       .HP_COUNTER
```

```
. . . . . .HP_D_TO_A
. . . . . .HP_MULTIMETER
. . . . . .HP_POWERMETER
. . . . .I1553_IO
. . . . . .PC
. . . . . .VAX
. . . . .I488_IO
. . . . . .PC
. . . . . .VAX
. . . . .INTERNAL_IO
. . . . . .PC
. . . . . .VAX
. . . . .RECORDING
. . . . . .PC
. . . . . .VAX
. . . . .REDUCTION
. . . . . .PC
. . . . . .VAX
. . . . .STIM_RESPONSE
. . . . . .PC
. . . . . .VAX
. . . . .TRAJECTORY
. . . . .USER
. . . . . .USER_LIB
. .LOCKHEED_POA2
. .SAIC3
. .SAICBB3
. .SAIC_COMSYS
. . .SOURCE
. . .USERMAN
. .SAIC_SGML
. .SAIC_TP2
. .SA_E
. .SCT_OP2
. . .DIJKSTRA
. . . .ALGORITHM
. . . .DEMONSTRATION
. . .DPA
. . . .ALGORITHM
. . . .DEMONSTRATION
. . .MASKING
. . . .ALGORITHM
. . . .DEMONSTRATION
. . .RET
. . . .ALGORITHM
. . . .DEMONSTRATION
. .SOFTECH2
. . .ADALIB
. .SRI
. . .ADALIB
. .UNISYS_ACE3
. . .DESIGN
```

```
.       .       .DOCS
.       .       .MISC
.       .       .READMES
.       .       .SRC
.       .UNISYS_RLF2
.       .       .ADAKNET
.       .       .ADATAU
.       .       .COMMON
.       .       .GADFLY
.       .       .RBDL
.       .       .SNDL
.       .WESTINGPR2_2
.       .       .VAXADALIB
.       .WESTING_NP
.       .       .ADALIB
.WESTING_PARSE
```